

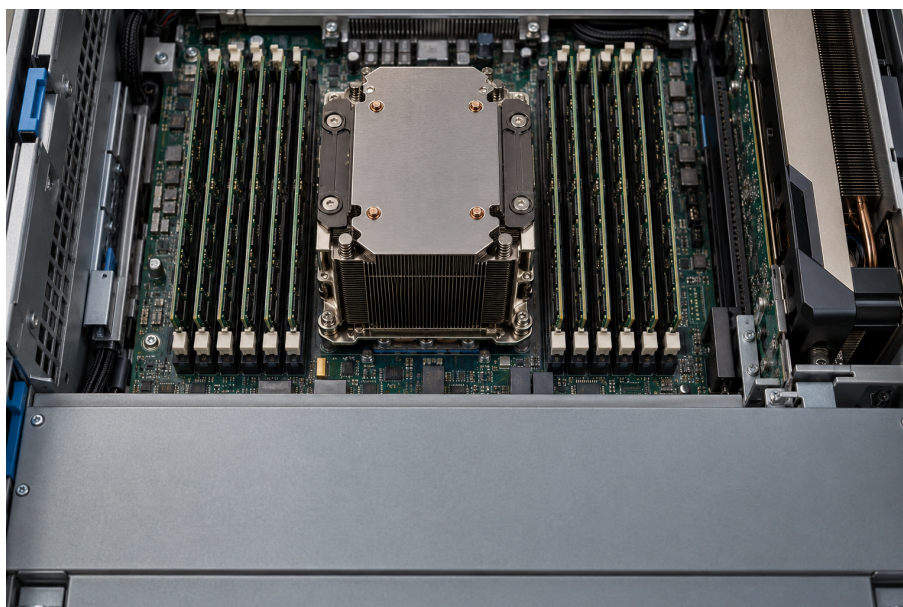


Kubernetes 1.37 Memory QoS: GPU Inference Tradeoffs

Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-10-30 · kubernetes memory qos · kubernetes 1.37 · gpu inference · cgroup v2 · memory throttling · memory protection · host memory · inference latency · oom

Original article: <https://gpusmith.com/articles/en/kubernetes-1-37-memory-qos-gpu-inference>



In this article

1. Executive Summary
 2. Introduction and Background
 3. Key Changes
 4. GPU Inference Host-Memory Inventory
 5. Implementation Considerations and Process Changes
 6. Canary Test Design
 7. Control Verification and Rollback
 8. Data Analysis and Evidence
 9. Implications and Future Directions
 10. Conclusion
-

Executive Summary

Kubernetes 1.37 Memory Quality of Service (QoS) governs host memory around graphics processing unit (GPU) inference. The official **September 14, 2026** announcement makes the feature gate Beta and enabled by default. (kubernetes.io) Its behavioral defaults remain inactive: `memoryThrottlingFactor: null` and `memoryReservationPolicy: None` write no new `memory.high`, `memory.min`, or `memory.low` settings. (raw.githubusercontent.com) An explicit factor carried from an older configuration can preserve throttling through upgrade. (kubernetes.io) NVIDIA distinguishes host dynamic random-access memory (DRAM) from memory attached to each GPU; these capacities require separate budgets. (docs.nvidia.com)

Operators therefore have four choices: retain the defaults, enable throttling, enable tiered protection, or combine both. Under **TieredReservation**, Guaranteed workloads receive hard reclaim protection through `memory.min`, Burstable workloads receive best-effort protection through `memory.low`, and BestEffort workloads receive neither. (raw.githubusercontent.com) (kubernetes.io) A configured `memory.high` introduces reclaim and allocation throttling rather than directly triggering the out-of-memory (OOM) killer. (docs.kernel.org) The operational question is whether that trade improves the application's service-level objective (SLO), measured under representative contention.

The important boundary is the node. Reservation applies to every pod there, and protected cgroup charges include file-backed page cache. A model-loading or file-reading neighbor therefore belongs in the experiment, even if it performs no GPU computation. (kubernetes.io) (docs.kernel.org) Host demand also depends on the serving stack: **Accelerate** can place weights across GPU memory, central processing unit (CPU) memory, and disk. (huggingface.co) **PyTorch** documents loading techniques intended to reduce memory and compute. (docs.pytorch.org) Device headroom alone cannot establish host-memory headroom.

The recommended decision method is an isolated, reversible canary. Compare identical loading and serving workloads across the policy alternatives, recording cgroup files, pressure stall information (PSI), OOM evidence, latency, useful throughput, and GPU activity. PSI measures stalled tasks, making it useful alongside occupancy. (docs.kernel.org) Keep the defaults when the evidence does not justify a policy change. Rollback

must inspect effective controls: existing containers can retain `memory.high` until restart or resize. (kubernetes.io) This report supplies a test design and declared arithmetic examples; it does not claim measured inference gains from Memory QoS.

Introduction and Background

Private inference clusters have two related capacity problems: fitting computation and model state on accelerators, and supplying the host memory needed around that execution. CUDA documents distinct CPU-attached and GPU-attached memory, while Unified Memory behavior can vary with hardware, operating system, kernel, and interconnect. (docs.nvidia.com) This report concerns the host side of that boundary. It uses **video random-access memory (VRAM)** to refer to device memory. **GPU high-bandwidth memory (HBM)**, device key-value (KV) caches, and accelerator allocation settings need their own measurements.

The distinction becomes practical during startup. Loading demand depends on how checkpoint storage is brought into physical memory. PyTorch's checkpoint recipe describes memory mapping that delegates physical-memory loading and unloading to the operating system. (docs.pytorch.org) That is a reason to examine the actual loader and its phases rather than estimate host demand from the final device allocation. Record cache state so that loading runs remain comparable.

As of October 30, 2026, the relevant Kubernetes decision is whether to configure the optional behaviors exposed by Memory QoS. The release-specific kubelet application programming interface (API) says that an unset throttling factor leaves `memory.high` unset, and defines tiered protection separately. (raw.githubusercontent.com) Turning a feature gate on and choosing those behaviors are distinct actions. The distinction also prevents a false diagnosis: observing memory pressure after an upgrade does not, by itself, establish that Memory QoS began writing controls.

For GPU Smith's audience of private-cluster operators, the useful deliverable is a reproducible capacity, SLO, and rollback decision. This report separates documented mechanisms from proposed experiments and hypothetical arithmetic. It does not prescribe a universal reservation percentage or assume a particular model server. Docker's operational guidance likewise calls for testing an application's memory requirements before production deployment. (docs.docker.com) The same discipline should extend to loaders, offload paths, companion services, and neighboring workloads.

Key Changes

A default-on gate with inactive behavioral defaults

The dated release announcement is the anchor for **1.37 behavior**: the gate is enabled, but operators must configure throttling or reservation to activate them. Its `null` representation corresponds operationally to omitting the factor; the release-specific API documents a `nil` default. (kubernetes.io) (raw.githubusercontent.com) The release defaulting code separately supplies `None` when the `reservation-policy` field is empty. (raw.githubusercontent.com)

An upgrade review should inspect the rendered kubelet configuration, including distribution templates and configuration-management overlays. The announcement explicitly preserves a configured factor across upgrade. (kubernetes.io) Consequently, two nodes running the same Kubernetes release can have different effective policies. Record configuration provenance as well as the release number.

The Kubernetes Enhancement Proposal (KEP) is a design and implementation reference, but its moving main branch includes later-version material. It now discusses additional reservation modes associated with **1.38**, which should not be treated as available choices in a **1.37** rollout. (github.com) Historical defaults in older sections also require version context. The current API and dated announcement provide the operational anchors.

Protection follows the workload's QoS class

Table 1 summarizes the documented policy mapping. It combines the kubelet API, pod quality-of-service (QoS) documentation, and kernel definitions. (raw.githubusercontent.com) (docs.kernel.org)

Workload or control	With tiered reservation	With an explicit throttling factor	Operator interpretation
Guaranteed	Requested memory gets <code>memory.min</code> .	No <code>memory.high</code> is assigned by this feature. (kubernetes.io)	Check the request, limit, and file-reading peak together.
Burstable	Requested memory gets <code>memory.low</code> .	A calculated <code>memory.high</code> applies. (kubernetes.io)	Soft protection can coexist with allocation throttling.
BestEffort	No request protection.	A calculated <code>memory.high</code> applies. (kubernetes.io)	Include these neighbors in contention tests.
<code>memory.min</code>	Hard reclaim protection within effective hierarchy bounds.	Independent of the throttling setting.	Overcommitted protection is not unlimited capacity. (docs.kernel.org)
<code>memory.low</code>	Best-effort reclaim protection.	Independent of the throttling setting.	Reclaim remains possible when needed. (freedesktop.org)
<code>memory.high</code>	Does not itself reserve memory.	Reclaim and throttling boundary.	A stall mechanism differs from a hard allocation limit. (freedesktop.org)

Guaranteed-pod warning: With `TieredReservation`, `memory.min` equals `memory.max` for Guaranteed pods. A large page cache can prevent sufficient reclaim before the hard limit is reached, resulting in an OOM kill. ([Kubernetes](https://kubernetes.io)) ([Linux kernel](https://linux.kernel.org)) Require limit headroom for simultaneous anonymous allocations and page cache, and include a Guaranteed model-loading or file-reading case in the canary.

The mapping makes requests an input to reclaim behavior, not merely an inventory field. Ancestors matter as well: effective protection generally requires corresponding allocations throughout the hierarchy. (freedesktop.org) Inspect the container, pod, QoS, and parent paths before concluding that a child value provides effective protection.

Throttling creates an intermediate boundary

The KEP calculates a Burstable container's threshold from its request plus the configured factor multiplied by the gap between limit and request, rounded down to the system page size. It substitutes node allocatable when the applicable limit is absent. (github.com) **A smaller factor moves this boundary toward the request;** that mathematical relationship does not establish a better production setting.

The kernel distinguishes this boundary from `memory.max`: high-boundary pressure invokes throttling and reclaim, while an unreclaimable hard-limit breach can invoke the cgroup OOM killer. (docs.kernel.org) Enabling throttling may therefore change the form of memory stress. Treat reduced kills, increased latency, or changed throughput as outcomes to measure, not automatic consequences.

GPU Inference Host-Memory Inventory

A useful inventory follows the deployment through initialization, warmup, ordinary serving, and overload. It should identify ownership and lifetime, not simply sum configured maxima. **Measure simultaneous demand:** a loading buffer that disappears before steady serving differs from a persistent offload allocation.

- **Checkpoint staging:** Record transient allocations and the loader path. PyTorch documents techniques for reducing checkpoint-loading memory and compute. (docs.pytorch.org)
- **Mapped weights and file cache:** Record mapped-file activity alongside resident anonymous memory. The kernel's cgroup accounting includes both anonymous memory and filesystem cache. (docs.kernel.org)
- **Pinned transfer buffers:** Inventory allocations used for asynchronous transfers. NVIDIA's best-practices guide requires pinned host memory for the asynchronous transfer path it describes and cautions against excessive pinning. (docs.nvidia.com)
- **CPU weight offload:** Record the per-device allocation and transfer path. **vLLM** expresses its CPU offload setting in gibibytes (GiB) per GPU and documents a default of **0**, meaning no offload. (docs.vllm.ai)
- **Processing caches:** Inventory multimodal preprocessing as well as model weights. vLLM identifies its processor cache as a CPU-memory tuning target. (docs.vllm.ai)
- **Companion services:** Measure tokenizers, routers, logging, metrics, and other sidecars independently, including their overlap with startup and request bursts.

The loader can materially alter this inventory. PyTorch's checkpoint recipe combines memory mapping, a metadata-only model construction path, and state assignment to reduce loading work and memory use. (docs.pytorch.org) This is an application-level opportunity to investigate before attributing every host-memory peak to inadequate node capacity. It is not proof that every serving framework uses that path.

Parallel loading deserves a separate test. vLLM documents a tensor-parallel loading path in which each process reads the whole model before splitting it. (docs.vllm.ai) A proposed canary should therefore include simultaneous worker startup and the normal rollout surge. Testing only an already warmed replica omits a potentially different memory and storage pattern.

Offload makes host capacity part of the steady serving path. Accelerate's automatic placement can fill GPU capacity before CPU memory and disk; it documents additional inference overhead when layers move through those locations. (huggingface.co) vLLM likewise cautions that CPU offload requires a fast CPU-to-GPU interconnect. (docs.vllm.ai) The correct experiment holds offload settings constant while changing the kubelet policy.

File-backed demand also needs careful interpretation. Accelerate documents memory-mapped disk placement when CPU memory is insufficient, while its sharded loading path bounds loading memory in relation to the largest shard. (huggingface.co) Those statements describe particular framework paths, not universal node sizing rules. Record the checkpoint format, loader, shard layout, storage source, and cache state with every result.

Finally, identify monitoring boundaries. Triton exposes total and used pinned-pool byte metrics, which can inform an inventory for deployments using that server. (docs.nvidia.com) They do not establish the accounting or reclaim behavior of every CUDA allocation. Reconcile application metrics with observed cgroup values instead of assuming that a framework pool metric equals total container memory.

Implementation Considerations and Process Changes

Verify eligibility and supported software

The KEP calls for a unified Linux cgroup-v2 hierarchy, a compatible Container Runtime Interface (CRI) implementation, and node-allocatable enforcement. Its prerequisite kernel is **5.9 or later**. (github.com) Verify the installed distribution's supported combination.

- **Hierarchy:** Confirm the actual mounted cgroup version and controller paths. Linux documents cgroup v2 as a unified hierarchy. (man7.org)
- **Kernel:** Record the build, vendor patches, and memory-pressure instrumentation alongside the KEP prerequisite. (github.com)
- **Runtime capability:** The KEP's historical feature prerequisites identify **containerd 1.6+** or **CRI-O 1.22+** for the needed unified support. (github.com)
- **Runtime support:** Do not equate that historical minimum with current deployment guidance. The current containerd matrix recommends **2.4.0+ or 2.3.0+** for Kubernetes **1.37**, using CRI v1. (containerd.io)
- **Reservation enforcement:** Verify that node allocatable, system reservations, and eviction headroom reflect the intended operating model. (kubernetes.io)
- **Runtime evidence:** Confirm that the runtime applies the requested controls. The Open Container Initiative (OCI) specification defines a unified map whose keys refer to cgroup files. (raw.githubusercontent.com)

CRI-O is another implementation of the CRI and documents its use of OCI-compatible runtimes. (cri-o.io) The acceptance criterion should be the installed stack's actual behavior and support matrix.

Choose the smallest justified policy change

The following are proposed decision categories, not performance rankings:

- **Retain defaults:** Appropriate when the current failure policy and SLO are satisfactory, or when the experiment has not established a benefit.
- **Throttling only:** Test whether earlier reclaim improves the desired failure behavior without unacceptable loading or serving stalls.
- **Tiered reservation only:** Test whether protecting requested memory improves the selected workload while preserving acceptable neighbor behavior.
- **Both behaviors:** Test their interaction after understanding each individually, especially during concurrent loading and contention.

These choices correspond to independently configurable throttling and reservation fields in the release-specific API. ([raw.githubusercontent.com](https://raw.githubusercontent.com/kubernetes/kubernetes/master/pkg/apis/kubelet/v1beta1/types.go)) Keep the rendered configuration under version control.

Explicit factor values must be **greater than 0 and no greater than 1** under release-specific validation. ([raw.githubusercontent.com](https://raw.githubusercontent.com/kubernetes/kubernetes/master/pkg/apis/kubelet/v1beta1/types.go)) A factor at the upper boundary changes where the high boundary is calculated; it does not remove the need to inspect effective settings. For the inactive baseline, omit the factor and leave reservation at None.

A proposed canary overlay, merged into the existing kubelet configuration, can test both behaviors as follows.

The factor is an experimental input, not a recommended default. The API and validation define these fields and the permitted factor range. ([raw.githubusercontent.com](https://raw.githubusercontent.com/kubernetes/kubernetes/master/pkg/apis/kubelet/v1beta1/types.go))

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
memoryThrottlingFactor: 0.9
memoryReservationPolicy: TieredReservation
```

Account for the entire node

The reservation policy is node-wide, so a dedicated canary pool is a useful boundary for limiting the experiment's exposure. (kubernetes.io) Label and isolate that pool using the cluster's existing scheduling practices, then include its actual daemon and companion workload mix.

Protecting a child without checking parents is also incomplete. The pod-QoS documentation describes ancestor settings for the root pod hierarchy and Burstable class; systemd's resource-control documentation independently explains corresponding ancestor allocations. (kubernetes.io) (freedesktop.org) A before-and-after ledger should cover both configuration intent and observed hierarchy state.

Eviction deserves a separate review. Kubernetes' Linux `memory.available` calculation excludes inactive file memory on the assumption that it is reclaimable. (kubernetes.io) Protection and reclaim behavior therefore need observation together with eviction signals. Do not treat a single free-memory chart as the complete explanation for a node's decisions.

Canary Test Design

Make the load comparison reproducible

The canary should change policy while keeping the application experiment stable. MLCommons defines benchmark scenarios through request arrival patterns and specified measurement metrics; that is a useful discipline even when the production test is not an MLPerf submission. (mlcommons.org) Keep a replayable workload description with the node image and configuration.

- **Artifact identity:** Fix model weights, tokenizer, serving image, and checkpoint layout.
- **Hardware identity:** Match CPU, DRAM, GPU, interconnect, storage, and node-image characteristics.
- **Request identity:** Fix input and output distributions, request count, and arrival process.
- **Concurrency:** Record both offered load and the active-request limit.
- **Application settings:** Freeze batching, parallelism, offload, and cache settings.
- **Startup phase:** Include cold loading and the expected simultaneous replica start.
- **Steady phase:** Run representative warmed traffic.
- **Contention phase:** Exercise the intended neighbor mix and file-reading activity.
- **Evidence window:** Align logs, counter deltas, pressure measurements, and request timestamps.

SGLang's benchmark documentation illustrates why arrival process and concurrency are separate controls: a finite rate uses a Poisson process, while maximum concurrency has its own setting. (docs.sglang.io) A comparison that changes both the kubelet policy and the offered-load pattern cannot isolate the policy's effect.

Define latency at the user-visible boundary. Ray distinguishes **time to first token (TTFT)**, from arrival to the first streamed token, from **time per output token (TPOT)** during decode. (docs.ray.io) Measure both when streaming matters, along with completion latency, errors, and useful throughput. A warmed average can conceal a loading regression or a pressured tail.

Exercise mixed-workload failure modes

These are proposed test cases, not reported incidents:

- **Large file reader:** Observe whether protected file-backed demand changes reclaim, pressure, or neighbor latency.
- **Concurrent model loader:** Check loading completion and the serving replica's SLO together.
- **Offload-heavy neighbor:** Hold transfer and offload settings fixed while measuring contention.
- **Unprotected neighbor:** Check whether the intended reclaim order remains acceptable for BestEffort or other unprotected work.
- **Daemon activity:** Include realistic logging, monitoring, and system-service peaks.

The file-reader case follows directly from documented cgroup file-cache accounting. (docs.kernel.org) The offload case follows from frameworks that deliberately use CPU memory or move layers through it. (huggingface.co) Success should mean acceptable node-wide outcomes, not merely an improved chart for the selected model server.

GPU telemetry can corroborate the timeline but needs precise labels. Ray's GPU cache-utilization metric describes KV-cache occupancy, not compute activity. (docs.ray.io) NVIDIA's Data Center GPU Manager (DCGM) profiling metrics are interval averages; its streaming-multiprocessor activity can include warps waiting on memory. (docs.nvidia.com) Therefore, do not label the complement of such a metric as exact application idle time. Use suitable traces when attribution matters.

Control Verification and Rollback

Keep a control ledger and an outcome ledger

Table 2 is a proposed evidence worksheet. It separates file settings from observed pressure and service outcomes. Docker's cgroup-v2 inspection guidance and the OCI unified-map specification support checking actual files rather than relying only on declared configuration. (docs.docker.com) (raw.githubusercontent.com)

Evidence	Baseline record	Canary record	Interpretation
Effective configuration	Gate, factor presence, reservation policy	Same fields plus intended change	Establish what was actually tested.
<code>memory.min</code> , <code>memory.low</code>	Container and ancestor values	Same paths after reconciliation	Check effective protection. (freedesktop.org)
<code>memory.high</code> , <code>memory.max</code>	Container values	Same values and changes	Separate throttling from the hard limit. (docs.kernel.org)
<code>memory.current</code> , <code>memory.stat</code>	Phase-specific occupancy	Anonymous and file components	Identify the type of charged demand. (docs.kernel.org)
<code>memory.events</code> , <code>memory.events.local</code>	Counter snapshots	Deltas over matching intervals	Separate high, max, OOM, and killed-process evidence. (docs.kernel.org)
<code>memory.pressure</code>	Matching windows and totals	Node and relevant cgroup values	Relate stalls to service outcomes. (docs.kernel.org)
Request and device metrics	Latency, useful throughput, errors, GPU activity	Same definitions and collection interval	Avoid treating device occupancy as useful work. (docs.ray.io)

Counter interpretation matters. Kernel event counters can include descendants, while local event files distinguish local scope; Linux manual documentation also notes mount-option effects on aggregation. (docs.kernel.org) (man7.org) Do not add parent and child counters as though they were independent events. CRI-O's monitoring path can report OOM situations, providing another timestamped view for deployments using it. (cri-o.io)

PSI supplies **some** stalls, affecting at least some tasks, and **full** stalls, affecting all non-idle tasks. Its documented recent windows are **10, 60, and 300 seconds**, with cumulative stall time in microseconds. (docs.kernel.org) Record which scope and interval each chart represents. Grafana Alloy's Unix exporter can expose host pressure data; containerized deployment requires the appropriate host mounts and configuration. (grafana.com)

Revert configuration and verify cleanup

A rollback plan must specify both configuration changes and the expected reconciliation path:

- **Reservation policy:** Restore None before reverting to the inactive baseline.
- **Throttling field:** Remove the explicit factor for that baseline.
- **Gate validation:** If disabling the gate, check configuration compatibility before restarting kubelet.
- **Ancestor cleanup:** Inspect the pod-root and Burstable protection files after startup.
- **Container cleanup:** Verify existing containers individually and restart or resize where the documented path requires it.

The documented startup cleanup clears root `memory.min` and `memory.low`, and Burstable-ancestor `memory.low`; child values can persist without effective ancestor protection. Existing `memory.high` can remain until restart or in-place resize. (kubernetes.io) Changing a configuration file alone is therefore insufficient rollback evidence.

Release-specific validation adds a practical detail: a disabled MemoryQoS gate allows an unset factor or the historical **0.9** value, while other explicit factors are rejected; tiered reservation also requires the gate. (raw.githubusercontent.com) The simpler rollback artifact omits the factor and selects None. Verify the resulting cgroup files before declaring recovery.

Data Analysis and Evidence

Compare policies without inventing results

Table 3 is an unfilled results template. Each cell is an instruction to measure, not a benchmark result. SGLang's distinction between total throughput and goodput, requests that succeed and meet the specified SLOs, motivates including useful throughput explicitly. (docs.sglang.io)

Policy arm	Cold-load evidence	Serving evidence	Contention and decision evidence
Defaults	Load time and host-memory peak	Tail latency, errors, goodput	Reference outcome and failure behavior
Throttling only	High-event deltas and loading stalls	TTFT, TPOT, pressure, goodput	Accept or reject against declared SLO
Tiered reservation only	File-cache and anonymous demand	Selected workload and neighbor latency	Protection benefit versus node-wide cost
Both	Combined control ledger	Same service metrics	Interaction and verified rollback

An acceptable outcome may differ by pool. A latency-oriented serving pool can choose a different failure tradeoff from a batch-loading pool. Write those decision criteria before viewing results, and preserve rejected runs as evidence. Systemd's pressure documentation links resource stalls with growing application latency, supporting correlation while leaving the workload-specific threshold to measurement. (systemd.io)

Quantile handling can otherwise invalidate the comparison. Prometheus warns against averaging precomputed summary quantiles across replicas; histogram-based estimates also depend on interpolation within buckets. (prometheus.io) Preserve request-level observations where practical, or use an aggregation and bucket scheme appropriate to the service SLO.

The release-specific kubelet source defines alpha gauges named `kubelet_memory_qos_node_memory_min_bytes` and `kubelet_memory_qos_node_memory_low_bytes`, tracking the respective Guaranteed and Burstable protection totals. (raw.githubusercontent.com) Treat these as configuration observability alongside file inspection, not evidence that an application SLO improved.

Declared calculations, not measured capacity claims

Hypothetical Example: Suppose a node has **512 GiB** physical DRAM, with **16 GiB** system reserve, **8 GiB** kube reserve, and **8 GiB** eviction headroom. These are analyst-selected assumptions. Allocatable is then **480 GiB**. Kubernetes and Red Hat documentation both describe reserving resources before deriving pod-available capacity. (kubernetes.io) (docs.redhat.com)

Assume **320 GiB** of protected requests and a measured unprotected peak entered as **96 GiB** for this hypothetical case. The residual is:

$$480 \text{ GiB} - 320 \text{ GiB} - 96 \text{ GiB} = 64 \text{ GiB}$$

The correct ledger is **allocatable minus protected requests minus the measured unprotected peak**, subtracting only additional reserves not already deducted. Starting from physical capacity instead requires subtracting the listed system, kube, and eviction amounts first. Subtracting those same reserves again from allocatable would double count. This arithmetic is a conservative planning envelope, not a model of effective kernel protection; hard and soft requests remain distinct and actual usage can differ.

Hypothetical Example: A request of **32 GiB**, limit of **64 GiB**, and factor **0.9** produce a pre-rounding high threshold of **60.8 GiB** using the KEP formula. These inputs are illustrative, not recommended. The actual value is rounded down to page size. (github.com) Neither a positive residual nor that threshold establishes acceptable latency.

For external context, MLCommons' **March 24, 2026** GPT-OSS benchmark specifies interactive **99th-percentile TTFT of at most 2.0 seconds** and **TPOT of at most 15 milliseconds**, versus **3.0 seconds** and **80 milliseconds** for its server scenario. (mlcommons.org) These are benchmark-specific objectives, not production defaults or measurements of Kubernetes Memory QoS.

Implications and Future Directions

The most useful rollout unit is a pool with a coherent host-memory and failure policy. This is an engineering recommendation derived from node-wide reservation scope, not a requirement to create a separate pool for every model. A mixed node should be evaluated as a shared capacity system: requests, real peaks, cache behavior, and acceptable neighbor degradation all matter.

A mature decision record should include the following acceptance rules:

- **Latency:** Accept only within the declared TTFT, TPOT, and completion-latency budgets.
- **Throughput:** Use successful, SLO-compliant work as well as aggregate tokens or requests.
- **Failure behavior:** State whether slower allocation, a rejected request, or a restarted replica is preferable under overload.
- **Neighbor impact:** Apply explicit budgets to protected and unprotected neighbors.
- **Operational recovery:** Require the effective-control ledger to demonstrate rollback.
- **Repeatability:** Re-run materially changed loaders, offload strategies, or node images.

Goodput's SLO-based definition supports the throughput rule, while the documented difference between high-boundary throttling and hard-limit OOM behavior supports making the failure preference explicit.

(docs.sglang.io) (docs.kernel.org) Neither mechanism supplies a universal business objective.

Monitoring should also be treated as part of the experiment. Red Hat advises testing PSI impact in a test environment or on one worker before broader enablement in latency-sensitive environments; that is general operational guidance, not a statement that its platform supports this upstream release. (docs.redhat.com) Ray provides a metrics-export path queryable through Prometheus, and Triton separately exposes GPU utilization and used-memory metrics. (docs.ray.io) (docs.nvidia.com) Preserve their definitions when integrating dashboards.

Future reservation choices do not remove the need for workload evidence. The current KEP contains later-version proposals, but this report's decision remains bounded to the documented Kubernetes 1.37 options. (github.com) Operators should reassess the implementation and support matrix when upgrading again. Until then, a successful canary establishes a conclusion for its tested configuration and workload envelope, with further testing warranted when either changes.

Conclusion

Enable explicit Memory QoS behavior only when the canary supports the intended host-memory tradeoff. The default-on gate does not, by itself, establish active throttling or reclaim protection. A configuration review and an effective-control ledger are the starting evidence; a release number alone cannot answer the operational question.

The decision should follow the workload. Inventory loading, file-backed demand, pinned buffers, offload, processing, and companion services across their actual lifetimes. Establish the node's usable memory budget without deducting the same reservations twice. Then compare the policy alternatives under identical loading, serving, and contention conditions.

The acceptance test belongs at the service and node boundaries together. Record user-visible latency, successful work, pressure, and failure behavior for the intended server and its neighbors. Interpret GPU telemetry according to its documented meaning. Keep loading regressions and neighbor costs visible even when the selected replica looks healthier.

Finally, make rollback demonstrable. Restore the intended configuration, verify ancestor protection, and check remaining container throttling values through the required reconciliation path. A reversible experiment should end with evidence of both the chosen steady state and recovery.

When those records show a worthwhile, repeatable benefit within the operator's SLO and failure policy, enable the tested behavior in a controlled expansion. When they do not, retaining the inactive defaults is a complete engineering decision. The report's contribution is that decision procedure, not a promised reduction in OOM events, lower latency, or higher GPU utilization.

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://kubernetes.io/blog/2026/09/14/kubernetes-v1-37-memory-qos-graduates-to-beta>
2. <https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.37/staging/src/k8s.io/kubelet/config/v1beta1/types.go>
3. <https://docs.nvidia.com/cuda/cuda-programming-guide/02-basics/understanding-memory.html>
4. <https://kubernetes.io/docs/concepts/workloads/pods/pod-qos>
5. <https://docs.kernel.org/admin-guide/cgroup-v2.html>
6. https://huggingface.co/docs/accelerate/usage_guides/big_modeling
7. https://docs.pytorch.org/tutorials/recipes/recipes/module_load_state_dict_tips.html
8. <https://docs.kernel.org/accounting/psi.html>
9. https://docs.docker.com/engine/containers/resource_constraints
10. <https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.37/pkg/kubelet/apis/config/v1beta1/defaults.go>
11. <https://github.com/kubernetes/enhancements/blob/master/keps/sig-node/2570-memory-qos/README.md>
12. <https://www.freedesktop.org/software/systemd/man/261/systemd.resource-control.html>
13. <https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html>
14. https://docs.vllm.ai/en/latest/configuration/engine_args
15. https://docs.vllm.ai/en/latest/configuration/conserving_memory
16. https://huggingface.co/docs/accelerate/concept_guides/big_model_inference
17. https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/metrics.html
18. <https://man7.org/linux/man-pages/man7/cgroups.7.html>
19. <https://containerd.io/releases>
20. <https://kubernetes.io/docs/tasks/administer-cluster/reserve-compute-resources>
21. <https://raw.githubusercontent.com/opencontainers/runtime-spec/main/config-linux.md>
22. <https://cri-o.io>
23. <https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.37/pkg/kubelet/apis/config/validation/validation.go>
24. <https://kubernetes.io/docs/concepts/scheduling-eviction/node-pressure-eviction>
25. <https://mlcommons.org/benchmarks/inference-datacenter>
26. https://docs.sglang.io/docs/developer_guide/bench_serving
27. <https://docs.ray.io/en/latest/serve/llm/benchmarks.html>
28. <https://docs.ray.io/en/latest/serve/llm/user-guides/observability.html>
29. <https://docs.nvidia.com/datacenter/dcgmlatest/learn/modules/profiling.html>
30. <https://docs.docker.com/engine/containers/runmetrics>

31. <https://grafana.com/docs/alloy/latest/reference/components/prometheus/prometheus.exporter.unix>
32. <https://systemd.io/PRESSURE>
33. <https://prometheus.io/docs/practices/histograms>
34. <https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.37/pkg/kubelet/metrics/metrics.go>
35. https://docs.redhat.com/en/documentation/openshift_container_platform/4.22/html/nodes/working-with-nodes
36. <https://mlcommons.org/2026/03/mlperf-inference-gpt-oss>

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the [hardware library](#), [GPU references](#), [networking references](#) and [vendor research](#). [Contact GPU Smith](#) with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: <https://gpusmith.com>

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.